Introduction to Formal Language Theory

Jeffrey Heinz

January 24, 2022

The material in this handout is covered in much greater detail in a number of textbooks including Harrison (1978); Hopcroft *et al.* (1979); Davis and Weyuker (1983); Hopcroft *et al.* (2001) and Sipser (1997). Here we will state definitions and theorems, but we will not cover the proofs of the theorems.

We begin with the following question: If we choose to model natural languages with formal languages, what kind of formal languages are they? We have some idea what natural languages are. After all, you are reading this! A satisfactory answer to answer this question however also requires being clear about what a formal language is.

1 Formal Languages

A formal language is a set of strings. Strings are sequences of symbols of finite length. The symbol Σ commonly denotes a finite set of symbols. There is a unique string of length zero, which is the empty string. This is commonly denoted with λ or ϵ.

A key operation on strings is *concatenation*. The concatenation of string x with string y is written xy. Note for all strings x, $x\lambda = \lambda x = x$. We can also concatenate two formal languages X and Y.

$$XY = \{xy : x \in X, y \in Y\}$$

If we concatenate a language with itself n times we write X^n. For example, $XX = X^2$. Finally for any language X, we define X^* as follows.

$$X^* = \{\lambda\} \cup X \cup X^2 \cup X^3 \ldots = \bigcup_{n \geq 0} X^n$$

where X^0 is defined as $\{\lambda\}$.

It follows that the set of all strings of finite length can be denoted Σ^*. Consequently formal languages can be thought of as subsets of Σ^**. How can we talk about such subsets?

2 Grammars

There are two important aspects to defining grammar formalisms. They are distinct, but related, aspects.

1. The grammar itself. This is an object and in order to be well-formed it has to follow certain rules and/or conditions.

2. How the grammar is associated with a language. A separate set of rules/conditions explains how a grammar *generates/recognizes/accepts* a language.

In other words, by itself, a grammar is more or less useless. But combined with a way to interpret it—a way to associate it with a formal language—it becomes a very powerful form of expression.

3 Regular Expressions

As a first example, consider regular expressions. These consist of both a syntax (which define well-formed regular expressions) and a semantics (which associate them unambiguously with formal languages). They are defined inductively.

Syntax		**Semantics**
REs include		
• each $\sigma \in \Sigma$		• $[\![\sigma]\!] = \{\sigma\}$
• ϵ		• $[\![\epsilon]\!] = \{\epsilon\}$
• $\varnothing$		• $[\![\varnothing]\!] = \{\}$
If R and S are REs then so are		
• (R·S)	*(concatenation)*	• $[\![(\mathrm{R}\cdot\mathrm{S})]\!] = [\![\mathrm{R}]\!]\cdot[\![\mathrm{S}]\!]$
• (R+S)	*(union)*	• $[\![(\mathrm{R}+\mathrm{S})]\!] = [\![\mathrm{R}]\!]\cup[\![\mathrm{S}]\!]$
• (R^*)	*(Kleene star)*	• $[\![(\mathrm{R}^*)]\!] = [\![\mathrm{R}]\!]^*$

We say a language is *regular* if there is a regular expression denoting it. The class of regular languages is denoted $[\![\mathrm{RE}]\!]$.

Exercise 1 *Write regular expressions for the following languages. Assume* $\Sigma = \{a, b, c\}$.

1. $\{a, \lambda\}$.

2. $\{w : w$ *contains the string* $aa\}$

3. $\{w : w$ *contains either the string* aa *or the string* $bb\}$

3.1 Generalized Regular Expressions

Syntax
GREs include

- each $\sigma \in \Sigma$
- ϵ
- $\varnothing$

Semantics

- $[\![\sigma]\!] = \{\sigma\}$
- $[\![\epsilon]\!] = \{\epsilon\}$
- $[\![\varnothing]\!] = \{\}$

If R and S are GREs then so are

- $(\mathrm{R}\cdot\mathrm{S})$ *(concatenation)*
- $(\mathrm{R}+\mathrm{S})$ *(union)*
- (R^*) *(Kleene star)*
- $(\mathrm{R}\&\mathrm{S})$ *(intersection)*
- $(\overline{\mathrm{R}})$ *(complement)*

- $[\![(\mathrm{R}\cdot\mathrm{S})]\!] = [\![\mathrm{R}]\!]\cdot[\![\mathrm{S}]\!]$
- $[\![(\mathrm{R}+\mathrm{S})]\!] = [\![\mathrm{R}]\!]\cup[\![\mathrm{S}]\!]$
- $[\![(\mathrm{R}^*)]\!] = [\![\mathrm{R}]\!]^*$
- $[\![(\mathrm{R}\&\mathrm{S})]\!] = [\![\mathrm{R}]\!]\cap[\![\mathrm{S}]\!]$
- $[\![\overline{\mathrm{R}}]\!] = \Sigma^* - [\![\mathrm{R}]\!]$

Theorem 1 $[\![RE]\!] = [\![GRE]\!]$

3.2 Cat-Union Regular Expressions

Syntax
CUEs include

- each $\sigma \in \Sigma$
- ϵ
- $\varnothing$

Semantics

- $[\![\sigma]\!] = \{\sigma\}$
- $[\![\epsilon]\!] = \{\epsilon\}$
- $[\![\varnothing]\!] = \{\}$

If R and S are CUEs then so are

- $(\mathrm{R}\cdot\mathrm{S})$ *(concatenation)*
- $(\mathrm{R}+\mathrm{S})$ *(union)*
- ~~(R^*) *(Kleene star)*~~
- ~~$(\mathrm{R}\&\mathrm{S})$ *(intersection)*~~
- ~~$(\overline{\mathrm{R}})$ *(complement)*~~

- $[\![(\mathrm{R}\cdot\mathrm{S})]\!] = [\![\mathrm{R}]\!]\cdot[\![\mathrm{S}]\!]$
- $[\![(\mathrm{R}+\mathrm{S})]\!] = [\![\mathrm{R}]\!]\cup[\![\mathrm{S}]\!]$
- ~~$[\![(\mathrm{R}^*)]\!] = [\![\mathrm{R}]\!]^*$~~
- ~~$[\![(\mathrm{R}\&\mathrm{S})]\!] = [\![\mathrm{R}]\!]\cap[\![\mathrm{S}]\!]$~~
- ~~$[\![\overline{\mathrm{R}}]\!] = \Sigma^* - [\![\mathrm{R}]\!]$~~

Exercise 2 *What kinds of formal languages do cat-union expressions describe?*

Theorem 2 $[\![CUE]\!] = \{L \subseteq \Sigma^* : |L| \text{ is finite}\} \subsetneq [\![RE]\!] = [\![GRE]\!]$

3.3 Star Free Expressions

Syntax
SFEs include

- each $\sigma \in \Sigma$
- ϵ
- $\varnothing$

Semantics

- $[\![\sigma]\!] = \{\sigma\}$
- $[\![\epsilon]\!] = \{\epsilon\}$
- $[\![\varnothing]\!] = \{\}$

If R and S are SFEs then so are

- $(\mathrm{R}\cdot\mathrm{S})$ *(concatenation)*
- $(\mathrm{R}+\mathrm{S})$ *(union)*
- ~~(R^*)~~ ~~*(Kleene star)*~~
- $(\mathrm{R}\&\mathrm{S})$ *(intersection)*
- $(\overline{\mathrm{R}})$ *(complement)*

- $[\![(\mathrm{R}\cdot\mathrm{S})]\!] = [\![\mathrm{R}]\!]\cdot[\![\mathrm{S}]\!]$
- $[\![(\mathrm{R}+\mathrm{S})]\!] = [\![\mathrm{R}]\!]\cup[\![\mathrm{S}]\!]$
- ~~$[\![(\mathrm{R}^*)]\!] = [\![\mathrm{R}]\!]^*$~~
- $[\![(\mathrm{R}\&\mathrm{S})]\!] = [\![\mathrm{R}]\!]\cap[\![\mathrm{S}]\!]$
- $[\![\overline{\mathrm{R}}]\!] = \Sigma^* - [\![\mathrm{R}]\!]$

Exercise 3 *Write star-free expressions for the following languages. Assume* $\Sigma = \{a, b, c\}$.

1. $\{a, \lambda\}$.
2. Σ^*.
3. $\{w : w$*contains the string* $aa\}$.
4. $\{w : w$*does not contain the string* $aa\}$.

Theorem 3 $[\![CUE]\!] \subsetneq [\![SFE]\!] \subsetneq [\![RE]\!] = [\![GRE]\!]$

4 Rewrite Grammars

There are many ways to define grammars which describe formal languages. Another influential approach has been rewrite grammars (Hopcroft *et al.*, 1979).

Definition 1 *A* rewrite grammar[1] *is a tuple* $\langle T, N, S, \mathcal{R}\rangle$ *where*

- $\mathcal{T}$ *is a nonempty finite alphabet of symbols. These symbols are also called the* terminal *symbols, and we usually write them with lowercase letters like* $a, b, c, \ldots$
- $\mathcal{N}$ *is a nonempty finite set of* non-terminal *symbols, which are distinct from elements of* $\mathcal{T}$*. These symbols are also called* category *symbols, and we usually write them with uppercase letters like* $A, B, C, \ldots$

[1] For a slightly different definition and some more description of rewrite grammars, see Partee *et al.* (1993, chap. 16).

- *S is the* start *category, which is an element of $\mathcal{N}$.*
- *A finite set of* production rules $\mathcal{R}$. *A production rule has the form*

$$\alpha \to \beta$$

where α, β belong to $(\mathcal{T} \cup \mathcal{N})^*$. In other words, α and β are strings of non-terminal and terminal symbols. While β may be the empty string we require that α include at least one symbol.

Rewrite grammars are also called *phrase structure grammars.*

Example 1 *Consider the following grammar G_1:*

- $\mathcal{T} = \{\textbf{\textit{john}}, \textbf{\textit{laughed}}, \textbf{\textit{and}}\}$*;*
- $\mathcal{N} = \{S,\ VP1,\ VP2\}$*; and*
- $$\mathcal{R} = \left\{ \begin{array}{l} S \to \textbf{\textit{john}}\ VP1 \\ VP1 \to \textbf{\textit{laughed}} \\ VP1 \to \textbf{\textit{laughed}}\ VP2 \\ VP2 \to \textbf{\textit{and laughed}} \\ VP2 \to \textbf{\textit{and laughed}}\ VP2 \end{array} \right\}$$

Example 2 *Consider the following grammar G_2:*

- $\mathcal{T} = \{a,\ b\}$*;*
- $\mathcal{N} = \{S,\ A,\ B\}$*; and*
- $$\mathcal{R} = \left\{ \begin{array}{l} S \to ABS \\ S \to \lambda \\ AB \to BA \\ BA \to AB \\ A \to a \\ B \to b \end{array} \right\}$$

Example 3 *Consider the following grammar G_3:*

- $\mathcal{T} = \{a,\ b\}$*;*
- $\mathcal{N} = \{S\}$*; and*
- $$\mathcal{R} = \left\{ \begin{array}{l} S \to ba \\ S \to baba \\ S \to bab \end{array} \right\}$$

The language of a rewrite grammar is defined recursively below.

Definition 2 *The (partial)* derivations *of a rewrite grammar* $G = \langle \mathcal{T}, \mathcal{N}, S, \mathcal{R} \rangle$ *is written* $D(G)$ *and is defined recursively as follows.*

1. The base case: S *belongs to* $D(G)$.
2. The recursive case: *For all* $\alpha \to \beta \in \mathcal{R}$ *and for all* $\gamma_1, \gamma_2 \in (\mathcal{T} \cup \mathcal{N})^*$, *if* $\gamma_1\alpha\gamma_2 \in D(G)$ *then* $\gamma_1\beta\gamma_2 \in D(G)$.
3. *Nothing else is in* $D(G)$.

Then the language of the grammar $L(G)$ *is defined as*

$$L(G) = \big\{ w \in \mathcal{T}^* : w \in D(G) \big\}.$$

Exercise 4 *How does* G_1 *generate* ***John laughed and laughed and laughed****?*

Exercise 5 *What language does* G_2 *generate?*

Exercise 6 *What language does* G_3 *generate?*

5 The Chomsky Hierarchy

"By putting increasingly stringent restrictions on the allowed forms of rules we can establish a series of grammars of decreasing generative power. Many such series are imaginable, but the one which has received the most attention is due to Chomsky and has come to be known as the Chomsky Hierarchy." (Partee *et al.*, 1993, p. 451)

Recall that rules are of the form $\alpha \to \beta$ with $\alpha, \beta \in (\mathcal{T} \cup \mathcal{N})^*$, with the further restriction that α was not the empty string.

Type 0 There is no further restriction on α or β.

Type 1 Each rule is of the form $\alpha \to \beta$ where α contains at least one symbol $A \in \mathcal{N}$ and β is not the empty string.

Type 2 Each rule is of the form $A \to \beta$ where $A \in \mathcal{N}$ and $\beta \in (\mathcal{T} \cup \mathcal{N})^*$.

Type 3 Each rule is of the form $A \to aB$ or $A \to a$ where $A, B \in \mathcal{N}$ and $a \in \mathcal{T}$.

There is one exception to the above restrictions for Types 1, 2 and 3. For these types, the production $S \to \lambda$ is allowed. If this production is included in a grammar then the formal language it describes will include the empty string. Otherwise, it will not.

To this we will add an additional type which we will call finite:

finite Each rule of is of the form $S \to w$ where $w \in \mathcal{T}^*$.

Type 0	recursively enumerable, computably enumerable
Type 1	context-sensitive
Type 2	context-free
Type 3	regular, right-linear[2]
finite	finite

Table 1: Names for classes of formal languages.

Each of these types goes by other names.

These names refer to both the *grammars* and the *languages*. These are different kinds of objects, so it is important to know which one is being referred to in any given context.

Theorem 4 (Chomsky Hierarchy) *1. The languages generated by type-3 grammars are a proper subset of the languages generated by type-2 grammars (Scott and Rabin, 1959).*

2. The languages generated by type-2 grammars are a proper subset of the languages generated by type-1 grammars (leaving aside the empty string) (Bar-Hillel et al.*, 1961).*

3. The languages generated by type-1 grammars are a proper subset of the languages generated by type-0 grammars.[3]

For details, see for instance Davis and Weyuker (1983).

Exercise 7 *1. Write a type 3 grammar which generates the language*

$$L = \{cv, cvcv, cvcvcv, \ldots\}.$$

Let's call this the "CV" language: the language where every word begins with a c*, ends with a* v*, and alternates* cs *and* vs*. This is an infinite language.*

2. Write a type 2 grammar which generates the language L=$\{\lambda$*, ab, aabb, aaabbb, . . .*$\}$*. This language is often called* $a^n b^n$ *since it begins with zero or more* as *which are followed by a matching number of* bs*. It's also an infinite language.*

3. Try to write a type 3 grammar for the language $a^n b^n$*. What problems do you run into?*

If we choose to model natural languages with formal languages, what kind of formal languages are they?

[2]Technically, right-linear grammars are defined as those languages where each rule is of the form $A \to aB$ or $A \to a$ where $A, B \in \mathcal{N}$ and $a \in \mathcal{T}$. Consequently is not possible for a right linear grammar to define a language which includes the empty string.

[3]This is a diagnolization argument of the kind originally due to Cantor. Rogers (1967) is a good source for this kind of thing.

References

Bar-Hillel, Y., M. Perles, and E. Shamir. 1961. On formal properties of simple phrase-structure grammars. *Zeitschrift fur Phonetik, Sprachwissenschaft, und Kommunikations-forschung* 14:143–177.

Davis, Martin D., and Elaine J. Weyuker. 1983. *Computability, Complexity and Languages*. Academic Press.

Harrison, Michael A. 1978. *Introduction to Formal Language Theory*. Addison-Wesley Publishing Company.

Hopcroft, John, Rajeev Motwani, and Jeffrey Ullman. 1979. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley.

Hopcroft, John, Rajeev Motwani, and Jeffrey Ullman. 2001. *Introduction to Automata Theory, Languages, and Computation*. Boston, MA: Addison-Wesley.

Partee, Barbara, Alice ter Meulen, and Robert Wall. 1993. *Mathematical Methods in Linguistics*. Dordrect, Boston, London: Kluwer Academic Publishers.

Rogers, Hartley. 1967. *Theory of Recursive Functions and Effective Computability*. McGraw Hill Book Company.

Scott, Dana, and Michael Rabin. 1959. Finite automata and their decision problems. *IBM Journal of Research and Development* 5:114–125.

Sipser, Michael. 1997. *Introduction to the Theory of Computation*. PWS Publishing Company.